

Why Scheme?

- Simple syntax
- Procedure syntax = data syntax
- Untyped
- No explicit pointer notation
- Automatic memory management
- Focus on higher-level concepts

Extending the language

| Concept | In Scheme |
|--|---------------------------------------|
| • Primitive expression | • Numbers, symbols |
| • Combination expression | • Lists, e.g., (fun arg1 arg2 ...) |
| • Abstraction (treat combination like a primitive) | • Define operator |

Scheme's top level

- Read one expression
- Evaluate that expression
- Print the value of that expression
- Repeat

(the read-eval-print loop)

Evaluation of primitive expressions

- Numbers, strings, etc., evaluate to themselves
- Symbols evaluate to values that have been assigned to them (usually by define)

```
(define x 12)
```

```
(define y "this is a string")
```

```
+
```

Evaluation of compound expressions (lists)

- Evaluate first element of list to obtain a procedure
- If normal procedure: evaluate remaining elements of list, then apply procedure to their values
- If special procedure: apply procedure to rest of list unevaluated

Observations

- Arithmetic operators are normal procedures
- The define operator is a special procedure
- Special procedures are called special forms
(also called syntactic forms)

Procedure definitions

```
(define (<name> <param1> <param2> ...)  
  <body>)
```

```
(define (square x) (* x x))  
(define (fourth-power x)  
  (square (square x)))  
(define (quadratic a b c x)  
  (+ a (* b x) (* c (square x)))))
```